Broadview
www.broadview.com.cn

碁峰
www.gotop.com.tw

博文视点原创精品大系

Java专家

Spring 技术手册

- Javaworld@TW技术论坛“技术手册”系列之一
- 以中文撰写的Spring入门书籍
- 完整项目详解，减少实作范例时摸索的时间
- 从实践中了解Spring AOP
- 项目实践：Spring在线书签

林信良 著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Spring 技术手册

林信良 著

電子工業出版社
Publishing House of Electronics Industry
北京 • BEIJING

内 容 简 介

本书为 Spring 的诸多概念提供了清晰的讲解，通过实际完成一个完整的 Spring 项目示例，展示了 Spring 相关 API 的使用，能够显著地减少每一位 Spring 入门者摸索 Spring API 的时间，并且从示例学习中获得提高。作者在写作之初，深入思考了每一位入门者所可能遇到的问题，通过简单的例子加上清晰的讲解，使得本书成为学习 Spring Web 开发的最佳读物。

本书内容全面深入，主要包括 Spring 入门、Bean/消息/事件、Spring 与面向方面编程（AOP）、JDBC 支持、Spring 与 Hibernate 的整合、Spring Web MVC 框架、View 层方案/Web 框架整合、远程/邮件/任务计划、Spring 在线书签完整项目示例等内容。

版权贸易合同登记号： 图字：01-2006-3106

图书在版编目（CIP）数据

Spring 技术手册 / 林信良著. —北京：电子工业出版社，2006.6
ISBN 7-121-02720-8

Ⅰ.S... Ⅱ.林... Ⅲ.计算机网络－程序设计－技术手册 Ⅳ.TP393-62

中国版本图书馆 CIP 数据核字（2006）第 058554 号

责任编辑：周 筠 魏 泉
印 刷：北京智力达印刷有限公司
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
经 销：各地新华书店
开 本：787×980 1/16 印张：25.25 字数：550 千字
印 次：2006 年 6 月第 1 次印刷
印 数：8 000 册 定 价：48.00 元（含光盘 1 张）

推荐序

对我这个专职从事教育训练的人来说，最重要的工作，莫过于寻找能够满足各种挑剔客户的优秀讲师，以及官方教材的编写者。因此，当我第一次从网络上浏览了"良葛格的学习笔记"网站之后，就开始对信良兄产生莫大的兴趣，不过当时他人并不在台湾，因而没有太多的交流。

直到 2005 年的 JavaTwo 大会，第一次有机会与他聊聊创作理念，以及对 Java 技术的看法之后，当下立即决定，把 SUN 教育训练中心的 Java 入门教材开发计划，全权委托他来实现。当教材开发完成，呈现在我眼前时，我发现，信良兄真是技术入门者的良伴，他真的用过心思，去了解这些技术的入门者，到底遇到了什么问题、想知道哪些知识，以及如何有效地引导初学者学习。所以，不管是之后邀请他教授 Java 课程，或是最近 SUN 官方 e-Learning 课程的设计，都是始终如一的质量优良。

Spring Framework 是近一两年，Java 社群对于 Enterprise Java 的应用上比较新的思考方向。相对于动辄吃掉好几百 MB 的 Applicaiton Server，Spring Framework 利用 IoC，DI，以及 AOP 的概念，创造出一个有别于 SUN 官方设计的轻巧架构，顿时获得 Java 社群的支持。当然，相对于整个 Java 的历史来说，Spring Framework 还很年轻，需要更多的时间，让大家去深入了解如何运用它，同时，也需要一本深入浅出的入门书籍，带领还没接触过 Spring Framework 的朋友，深入了解这个新趋势。我相信，信良兄会是写这本书的最佳人选，从内容的字里行间，可以确定他投注了很多的心血，思考每位入门者会遇到的问题，而这些问题，绝对可以在书中找到令读者满意的答案。如果您还不了解 Spring Framework，那么建议您直接拿起这本书，到柜台结账去吧，至于已经把玩过 Spring Framework 的朋友，建议您翻翻这本书，搞不好能找到您过去苦思不得其解的问题的答案，相信您也会带着这本书，愉快地走向柜台买下它。

王森

SUN 教育训练中心经理

2006 年 2 月 5 日

序

接触 Spring 是两年前的事了，当时有关 Spring 的文件还不是很多，凭借着零星的文件，以及网络上搜寻来的文件开始学习 Spring，并撰写编写了一些笔记文件来记录所得。

虽然说是学习 Spring 的使用，其实更多的时候是为了在学习程序设计时，考虑应当如何实现与组合各种架构，从 Spring 的使用中，可以了解许多从文件或书籍中所无法体会的抽象架构与观念。

随着两年来的经验与所学的增长，回过头去看先前为了学习 Spring 而留下的笔记文件，发现当中有许多的内容，现在又更加体会其作用，而现在有关 Spring 的文件与书籍已经不少，从这些新的文件与书籍中可以学习更多，体会更多。

“从实作实践中学习”一向是我最喜爱的学习方式之一，Spring 中许多的功能在实作实践时可以带给学习者从实作实践中学习的感觉，然而关于 Spring 中众多的 API（Application Interface）该如何使用的这个问题，的确让许多入门 Spring 的新手无所适从，并常花上大量的时间摸索，因而作为一本 Spring 入门书，在书中皆以完整的程序实作实践来示范如何使用 Spring，希望可以减少入门者学习 Spring 时的障碍与摸索。

林信良

2006 年 1 月 7 日

导　读

这份导读让您可以更了解如何使用本书。

本书许多的范例示范都使用完整的程序实践来展现，当您看到以下的程序代码示范时：

SpringDemo　　HelloBean.java

```
package onlyfun.caterpillar;

public class HelloBean {
    private String helloWord;

    public void setHelloWord(String helloWord) {
        this.helloWord = helloWord;
    }

    public String getHelloWord() {
        return helloWord;
    }
}
```

范例开始的左边名称为 SpringDemo，表示可以在光盘中找到对应的 SpringDemo 项目，而右边名称为 HelloBean.java，表示可以在项目中找到 HelloBean.java 文件。

如果使用以下的程序代码呈现，表示它是一个完整的程序内容，但不是项目的一部份，主要是用来展现一个完整的文件如何编写：

```
helloBean.class=onlyfun.caterpillar.HelloBean
helloBean.helloWord=Welcome
```

如果使用以下的程序代码呈现，则表示它是个程序代码片段，主要展现程序编写时需要特别注意的片段：

```
...
```

```
<beans>
   <bean id="beanFactoryModifier"
         class="onlyfun.caterpillar.SomeClass"/>
   ...
</beans>
...
```

由于书籍页面宽度的限制，程序代码中有时会因为过长而无法容纳于书籍的一行之中，此时不得不隔行表示时，会使用“→”符号，表示两行实际上是必须连接在一起的，例如：

```
...
   <bean id="configBean"
        class="org.springframework.beans.factory.
              → config.PropertyPlaceholderConfigurer">
      <property name="location">
         <value>hello.properties</value>
      </property>
   </bean>
...
```

在上面的程序片段中，实际上"class"属性的设定中，完整的程序代码编写是org.springframework.beans.factory.config.PropertyPlaceholder- Configurer。

在本书中会出现以下的对话框：

良葛格的话匣子<<<

在这个对话框中提供的可能是良葛格的一些主观性较强的意见、经验或建议，在这个对话框中也有可能是一些主题外的常识，或是提供一些额外的参考文件。

在范例文件的提供上，提供有 Eclipse 项目，而在 Web 应用程序的部分，则提供有 WAR（Web Application Archive）文件，可以直接将 WAR 文件放到 Servlet 容器的 webapps 目录下，容器启动后会自动解开文件并完成部署。

目 录

1 认识 Spring

Spring 的核心是个轻量级（Lightweight）的容器（Container），它是实现 IoC（Inversion of Control）容器、非侵入性（No intrusive）的框架，并提供 AOP（Aspect-oriented programming）概念的实现方式，提供对持久层（Persistence）、事务（Transaction）的支持，提供 MVC Web 框架的实现，并对一些常用的企业服务 API（Application Interface）提供一致的模型封装，是一个全方位的应用程序框架（Application framework），除此之外，对于现存的各种框架（Struts、JSF、Hibernate 等），Spring 也提供了与它们相整合的方案。

对于打算入门 Spring 的初学者来说，面对轻量级、非侵入性、容器、IoC、AOP 等术语往往不知所措，在这个章节中，将针对这些术语作一些介绍，以作为往后各个章节中，介绍 Spring 对这些术语概念的实现时的基础。

1.1 术语介绍

Spring 的目标之一，是作为一个全方位的应用程序框架（Application framework），由于在今日，应用程序在各个领域的应用越来越复杂，因而各个领域在设计与实践上的概念与术语也越来越多，作为一个全方位的应用程序框架（Framework），在使用 Spring 时就得接触到例如轻量级、容器、非侵入性（No intrusive）、IoC（Inversion of Control）、AOP（Aspect-oriented programming）等术语，对于入门 Spring 的初学者而言，以下的说明可以让您对这些术语与概念有个基本的认识。

- **轻量级**（Lightweight）

  轻量级的形容是相对于一些重量级的容器（如 EJB 容器）来说的，Spring 的核心包在文件容量上只有不到 1MB 的大小，而使用 Spring 核心包所需要的资源负担也是很小的，您甚至可以在小型设备中使用 Spring 的核心包。

- **非侵入性**（No intrusive）

  框架原来的用意是提供一个架构的实现，让开发人员可以在基于框架的基础上，快速地开发出遵循架构的所需的应用程序，然而有些框架一旦被使用，应用程序就与框架发生了依赖，例如大量使用了框架的 API，或直接继承 API 的某些类型等，都会使应用程序组件与框架发生依赖，而无法从框架中独立出来，更别说当中的组件可以直接重用到另一个应用程序之中。

  Spring 的目标之一是实现一个非侵入性（No intrusive）框架，希望让应用程序几乎感受不到框架的存在，减低应用程序在框架移植时的负担，进一步增加应用程序组件的可重用性（Reusability），简单地说，使用 Spring 的话，应用程序中某些组件可以直接拿到另一个应用程序或框架之中直接使用。

- **容器**（Container）

  Spring 提供容器功能，容器可以管理对象的生命周期、对象与对象之间的依赖关系，您可以使用一个配置文件（通常是 XML），在上面定义好对象的名称、如何产生（Prototype 方式或 Singleton 方式）、哪个对象产生之后必须设定成为某个对象的属性等，在启动容器之后，所有的对象都可以直接取用，不用编写任何一行程序代码来产生对象，或是建立对象与对象之间的依赖关系。

  换个更直白点的说明方式：容器是一个 Java 所编写的程序，原先必须自行编

写程序以管理对象关系，现在容器都会自动帮您作好。

- IoC（Inversion of Control）

  Spring 最重要的核心概念是 Inversion of Control，中文常译为“控制反转”，更具体的另一个名词是 Dependency Injection，中文常译为“依赖注入”；使用 Spring，您不必自己在程序代码中维护对象的依赖关系，只需在配置文件中加以设定，Spring 核心容器会自动根据配置将依赖注入指定的对象，在下一个小节当中，还会详细介绍 Inversion of Control 与 Dependency Injection 的概念。

- AOP（Aspect-oriented programming）

  Spring 最被人重视的另一方面是支持 AOP（Aspect-oriented programming）的实现，然而 AOP 框架只是 Spring 支持的一个子框架，说 Spring 框架是 AOP 框架并不是一个适当的描述，人们对于 AOP 的关注反映至 Spring 上，使得人们对于 Spring 的关注集中在它的 AOP 框架上，虽然有所误解，但也突显了 Spring 的另一个令人关注的特色。

  举个实际的例子来说明 AOP 的功能之一，假设您有个日志（Logging）的需求，您可以无须修改任何一行程序代码，就可以将这个需求加入至原先的应用程序之中，而若您愿意，也可以在不修改任何程序的情况下，将这个日志的功能移除。

  Spring 的 IoC 容器功能与 AOP 功能的实现是其重心所在，在 Spring 下实现了持久层、MVC Web 框架以及各种企业服务的 API 封装，它们的实现有些依重于 Spring 的 IoC 容器与 AOP 功能，Spring 的这些子框架或封装的 API 功能彼此可以独立，也可以结合其它的框架方案加以替代，Spring 希望提供 one-stop shop 的框架整合方案。

- 持久层

  Spring 提供对持久层的整合，如对 JDBC 的使用加以封装与简化，提供事务（Transaction）管理功能，对于 O/R Mapping 工具（Hibernate、iBATIS）的整合，Spring 也提供了解决的方案。

- Web 框架

  Spring 也提供 MVC Web 框架的解决方案，使用 Spring Web 框架的好处是可以善用 IoC 与 AOP 的功能，您甚至可以轻松地替换使用不同的 View 层技术，例如使用 JSP、结合 Tiles、使用 PDF 作为展现给使用者的画面技术。

也可以将自己所熟悉的 Web 框架与 Spring 整合，例如 Struts、JSF 等，都可以与 Spring 整合，而适用于当前所进行的应用程序。

- **其它企业服务的封装**

  对于一些服务，例如 JNDI、Mail、任务计划（Scheduling）、远程（Remoting）等，Spring 不直接提供实现，而是采取抽象层方式对这些服务进行封装，让这些服务在使用时可以有一致的使用模型，并且在使用上更为简化。

良葛格的话匣子<<<

如果您没有使用过其它框架的经验，只是希望藉由 Spring 的学习来吸收前人开发的经验，面对以上这些术语我想可能还是不太能了解，在 Spring 中这些概念都有具体的实现，您可以在往后的章节从实践中体会（事实上我也是如此）。

1.2　控制反转（Inversion of Control）

Spring 的核心概念是 IoC，IoC 的抽象概念是“依赖关系的转移”，转移是相对于过去不良的应用程序设计来说的，例如“高层模块不应该依赖低层模块，而是模块都必须依赖于抽象”是 IoC 的一种表现，“实现必须依赖抽象，而不是抽象依赖实现”也是 IoC 的一种表现，“应用程序不应依赖于容器，而是容器服务于应用程序”也是 IoC 的一种表现。

一连串抽象的文字轰炸之后，到底 IoC 是什么意思？IoC 全名 Inversion of Control，如果中文硬要翻译过来的话，就是“控制反转”。初看 IoC，从字面上不容易了解其意义，要了解 IoC，可先从 Dependency Inversion 开始了解，也就是依赖关系的反转，看看过去有哪些不良的依赖关系设计，而又要如何进行改进。

简单地说，在进行模块设计时，高层的抽象模块通常是与业务逻辑（Business logic）相关的模块，它应该具有重用性，而不依赖于低层的实现模块。例如低层模块可能是与硬件相关的软盘存取设计，而高层模块是个存盘备份的程序需求，如果高层模块直接执行低层模块的函数，就对低层模块产生了依赖关系。

在图 1.1 中，由于应用程序需要储存需求时直接执行了 `saveToFloppy()`，导致高层应用程序依赖于低层模块的 API，假设今天将应用程序移植至另一个平台上，而该平台使用 USB 磁盘存储，则这个应用程序无法直接重用，您必须加以修改才行，在这个例子中，由于低层模块的存储介质变更，造成了高层模块也必须跟着变更，这不是一个好的设计方式，

在设计上希望模块都依赖于模块的抽象，这样才可以重用高层的应用程序设计。

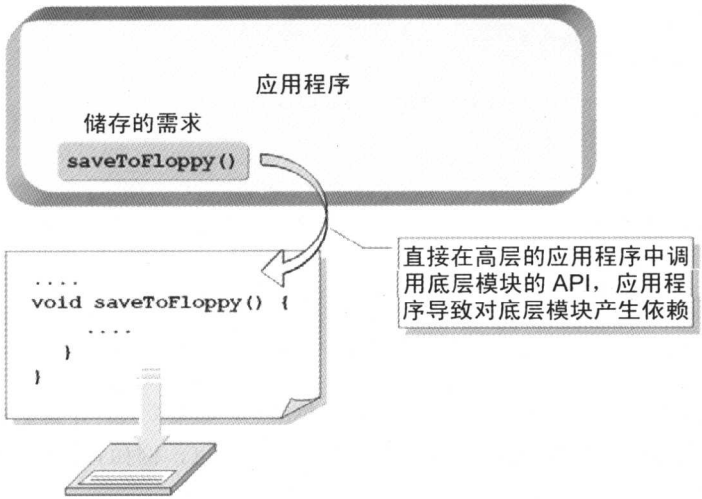

图 1.1 高层模块依赖低层实现

如果以对象导向的方式来设计，依赖反转（Dependency Inversion）的解释为“程序不应依赖实现，而是依赖于抽象接口”。以下面这个 Java 程序为例：

```
public class Business {
    private FloppyWriter writer = new FloppyWriter();
    ....
    public void save() {
        ...
        writer.saveToFloppy();
    }
}
```

Business 类的设计中，存盘的需求依赖于实际的 FloppyWriter 对象，如果今天想要将储存介质改为 USB 磁盘，则必须修改 Business 的程序，您无法直接重复使用 Business 类。

如果透过接口的声明，可以改进这种情况，例如可先定义一个 IDeviceWriter 接口：

```
public interface IDeviceWriter {
    public void saveToDevice();
}
```

接着所设计的 Business 类，在遇到存盘需求时，可以设计为依赖于 IDeviceWriter 接口，

而不是依赖于实际的 FloppyWriter，例如：

```
public class Business {
    private IDeviceWriter writer;

    public void setDeviceWriter(IDeviceWriter writer) {
        this.writer = writer;
    }

    public void save() {
        ....
        writer.saveToDevice();
    }
}
```

在这样的设计下，Business 类就是可以重用的，如果今天有存储至 Floppy 或 USB 磁盘的需求，只要针对这两种储存需求分别实现接口即可，例如针对 Floppy 存储设计一个 FloppyWriter 类：

```
public class FloppyWriter implement IDeviceWriter {
    public void saveToDevice() {
        ....
        // 实际储存至 Floppy 的程序代码
    }
}
```

或是针对 USB 磁盘存储设计一个 UsbDiskWriter 类：

```
public class UsbDiskWriter implement IDeviceWriter {
    public void saveToDevice() {
        ....
        // 实际储存至 UsbDisk 的程序代码
    }
}
```

如果应用程序需要 Floppy 存储的话，可以编写一个配置程序如下：

```
Business business = new Business();
business.setDeviceWriter(new FloppyWriter());
business.save();
```

同样的，如果应用程序需要 USB 磁盘存储的话，可以编写一个配置程序如下：

```
Business business = new Business();
```

```
business.setDeviceWriter(new UsbDiskWriter());
business.save();
```

可以看到，无论低层的存储如何变动，对于 Business 类来说都无需任何修改，您也可以编写一个配置管理程序，藉由一个简单的 XML 或是.properties 文件来更改配置，如此连上面的配置程序都不必编写，就可以简单地移植 Business 类，而事实上，Spring 核心容器就提供这样的配置管理的功能。

从上面的介绍来看，Dependency Inversion 的意思即是“程序不依赖于实现，而是程序与实现都要依赖于抽象”。

IoC 的 Control 是控制的意思，其实背后的意义也是一种依赖关系的转移，如果 A 依赖于 B，其意义即是 B 拥有控制权，您想要转移这种关系，所以依赖关系的反转即是控制关系的反转，将控制权由实现的一方转移至抽象的一方，藉由让抽象方拥有控制权，可以获得组件的可重用性，在上面的 Java 程序中，整个控制权从实际的 FloppyWriter 转移至抽象的 IDeviceWriter 接口上，而让 Business 依赖于 IDeviceWriter 接口，且 FloppyWriter、UsbDiskWriter 也依赖于 IDeviceWriter 接口。

程序的业务逻辑部分应该要设计为可以重用的，不应受到所使用框架或容器的影响，这样将来才有可能转移整个应用程序的业务逻辑至其它的框架或容器，如果业务逻辑过于依赖容器，则转移至其它的框架或容器时，就会发生困难。IoC 在容器的角度，可以用这么一句好莱坞名言来代表：“Don't call me, I'll call you.”以程序的术语来说的话，就是“不要向容器要求所需要的（对象）资源，容器会自动将这些对象给您！”。

IoC 要求的是容器不应该（或尽量不要）侵入应用程序，也就是不应出现与容器相依的 API，应用程序本身可以依赖于抽象的接口，容器可以透过这些抽象接口将所需的资源注入至应用程序中，应用程序不向容器主动要求资源，故而不会依赖于容器的特定 API，应用程序本身不会意识到正被容器使用，可以随时从容器系统中脱离，转移至其它的容器或框架而不用作任何的修改。

1.3 依赖注入（Dependency Injection）

IoC 模式基本上是一个高层的模式概念，在 Martin Fowler 的 Inversion of Control Containers and the Dependency Injection pattern（http://www. martinfowler.com/articles/ injection.html）中谈到，实现 IoC 有两种方式：Dependency Injection 与 Service Locator，Spring 所采用的是 Dependency Injection 来实现 IoC，中文翻译为依赖注入。

依赖注入的意义是："保留抽象接口，让组件（Component）依赖于抽象接口，当组件要与其它实际的对象发生依赖关系时，藉过抽象接口来注入依赖的实际对象。"

依赖注入在 Martin Fowler 的文章中谈到了三种实现方式：Interface injection、Setter injection 与 Constructor injection。并分别称其为 Type 1 IoC、Type 2 IoC 与 Type 3 IoC。

- **Type 2、Type 1**

在前一个小节中所提到的 Business 所实现的是 Type 2 IoC，透过 Setter（也就是 `setXXX` 方法）注入所依赖的对象，而 Type 3 IoC，则是在建构式上注入依赖关系，例如：

```
public class BusinessObject {
   private IDeviceWriter writer;

   public BusinessObject(IDeviceWriter writer) {
      this.writer = writer;
   }

   public void save() {
      ....
      writer.saveToDevice();
   }
}
```

Spring 鼓励使用 Setter injection（也就是 Type 2 IoC），但也允许您使用 Constructor injection，要使用 Setter 或是 Constructor 来完成依赖关系注入完全视您的需求而定，使用建构方法注入依赖对象的好处之一是，可以在建构对象的同时，一并完成依赖关系的建立，然而如果要建立的对象关系很多，则必须在建构式上宣告一长串的参数，这时使用 Setter 来注入依赖关系会是个不错的选择，因为 Setter 有明确的方法名称可用于了解注入对象的作用或类型，使用 `setXXX()` 这样的名称会比必须记忆或查询建构方法上某个参数位置所代表的对象要来得好。

- **Type 1**

Type 1 IoC 是 Interface injection，使用 Type 1 IoC 时会要求实作接口，对象所在的容器也会使用这个接口，容器知道接口上所规定的方法，所以可呼叫实作接口的对象来完成依赖关系的注入，例如容器的 API 中声明一个 `IDependency`：

```
public interface IDependency {
   public void createDependency(Map dependObjects);
}
```